Practical : 2

```
>import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import tensorflow as tf
from keras.models import Sequential
from keras.datasets import mnist
import keras
>(x_train,y_train),(x_test,y_test)=mnist.load_data()
x_train=x_train/255
x_test=x_test/255
>x_train
>x_train.shape
>x_test.shape
>model=Sequential()
model.add(keras.layers.Flatten(input_shape=(28,28)))
model.add(keras.layers.Dense(256,activation='relu'))
model.add(keras.layers.Dense(10,activation='softmax'))
>model.compile(optimizer='sgd',loss='sparse_categorical_crossentropy',metrics=['accuracy'])
H=model.fit(x_train,y_train,validation_data=(x_test,y_test),epochs=10)
>test_loss,test_acc=model.evaluate(x_test,y_test)
print("Loss=%.3f"%test_loss)
print("Accuracy=%.3f"%test_acc)
>import random
n=random.randint(0,9999)
plt.imshow(x_test[n])
plt.show()
pred=model.predict(x_test)
print("Digit is : " ,np.argmax(pred[n]))
>plt.imshow(x_test[2])
plt.show()
pred=model.predict(x_test)
print("Digit is : " ,np.argmax(pred[2]))
>plt.plot(H.history['accuracy'])
plt.plot(H.history['loss'])
plt.title('Training Loss and Accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Accuracy', 'Loss'])
>plt.plot(H.history['val_accuracy'])
plt.plot(H.history['val_loss'])
plt.title('Testing Loss and accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['Accuracy', 'Loss'])
```

Practical : 3

```
>import numpy as np
import pandas as pd
import random
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Flatten, Conv2D,Dense,MaxPooling2D
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.datasets import mnist
>(X_train, y_train),(X_test, y_test) = mnist.load_data()
>print(X_train.shape)
>X_train[0].min(),X_train[0].max()
>X_train = (X_train - 0.0) / (255.0 - 0.0)
X_test = (X_test - 0.0) / (255.0 - 0.0)
X_train[0].min(), X_train[0].max()
(0.0, 1.0)
>def plot_digit(image, digit, plt, i):
 plt.subplot(4, 5, i + 1)
 plt.imshow(image, cmap=plt.get_cmap('gray'))
 plt.title(f"Digit: {digit}")
 plt.xticks([])
 plt.yticks([])
plt.figure(figsize=(16, 10))
for i in range(20):
 plot_digit(X_train[i], y_train[i], plt, i)
plt.show()
>X_train = X_train.reshape((X_train.shape+ (1,)))
X_test = X_test.reshape((X_test.shape+(1,)))
>y_train[0:20]
>model = Sequential([
 Conv2D(32,(3,3), activation="relu",input_shape=(28,28,1)),
 MaxPooling2D((2,2)),
 Flatten(),
 Dense(100,activation="relu"),
 Dense(10,activation="softmax")])
>optimizer = SGD(learning_rate=0.01, momentum=0.9)
model.compile(
 optimizer=optimizer,
 loss="sparse_categorical_crossentropy",
 metrics=["accuracy"])
>model.summary()
>Model_log = model.fit(
    X_train,
    y_train,
    epochs=10,
    batch_size=15,
    verbose=1  )
>plt.figure(figsize=(16, 10))
```

```
for i in range(20):
 image = random.choice(X_test).squeeze()
 digit = np.argmax(model.predict(image.reshape((1, 28, 28, 1)))[0], axis=-1)
 plot_digit(image, digit, plt, i)
plt.show()
>predictions = np.argmax(model.predict(X_test),axis=-1)
accuracy_score(y_test,predictions)
>n = random.randint(0,9999)
plt.imshow(X_test[n])
plt.show()
>predicted_value = model.predict(X_test)
print("Handwritten number in the image is = %d" %np.argmax(predicted_value[n]))
>score = model.evaluate(X_test,y_test,verbose=0)
print('Test loss:' , score[0])
print('Testaccuracy:',score[1])
```

Practical : 4

```
>import pandas as pd
import numpy as np
import tensorflow as tf
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, recall_score, accuracy_score, precision_score
RANDOM_SEED = 2021
TEST_PCT = 0.3
LABELS = ["Normal","Fraud"]
>dataset = pd.read_csv("creditcard.csv")
>print("Any nulls in the dataset",dataset.isnull().values.any())
print('-------')
print("No. of unique labels",len(dataset['Class'].unique()))
print("Label values",dataset.Class.unique())
print('-------')
print("Break down of Normal and Fraud Transcations")
print(pd.value_counts(dataset['Class'],sort=True))
>count_classes = pd.value_counts(dataset['Class'],sort=True)
count_classes.plot(kind='bar',rot=0)
plt.xticks(range(len(dataset['Class'].unique())),dataset.Class.unique())
plt.title("Frequency by observation number")
plt.xlabel("Class")
plt.ylabel("Number of Observations")
normal_dataset = dataset[dataset.Class == 0]
fraud_dataset = dataset[dataset.Class == 1]
>normal_dataset = dataset[dataset.Class == 0]
fraud_dataset = dataset[dataset.Class == 1]
bins = np.linspace(200,2500,100)
plt.hist(normal_dataset.Amount,bins=bins,alpha=1,density=True,label='Normal')
plt.hist(fraud_dataset.Amount,bins=bins,alpha=0.5,density=True,label='Fraud')
plt.legend(loc='upper right')
plt.title("Transcation Amount vs Percentage of Transcaions")
plt.xlabel("Transcation Amount (USD)")
plt.ylabel("Percentage of Transcations")
plt.show()
>dataset
>sc = StandardScaler()
dataset['Time'] = sc.fit_transform(dataset['Time'].values.reshape(-1,1))
dataset['Amount'] = sc.fit_transform(dataset['Amount'].values.reshape(-1,1))
>raw_data = dataset.values
labels = raw_data[:,-1]
data = raw_data[:,0:-1]
train_data,test_data,train_labels,test_labels = train_test_split(data,labels,test_size =
0.2,random_state =2021)
>min_val = tf.reduce_min(train_data)
max_val = tf.reduce_max(train_data)
train_data = (train_data - min_val) / (max_val - min_val)
test_data = (test_data - min_val) / (max_val - min_val)
train_data = tf.cast(train_data,tf.float32)
```

```
test_data = tf.cast(test_data,tf.float32)
>train_labels = train_labels.astype(bool)
test_labels = test_labels.astype(bool)
normal_train_data = train_data[~train_labels]
normal_test_data = test_data[~test_labels]
fraud_train_data = train_data[train_labels]
fraud_test_data = test_data[test_labels]
print("No. of records in Fraud Train Data=",len(fraud_train_data))
print("No. of records in Normal Train Data=",len(normal_train_data))
print("No. of records in Fraud Test Data=",len(fraud_test_data))
print("No. of records in Normal Test Data=",len(normal_test_data))
>nb_epoch = 50
batch_size = 64
input_dim = normal_train_data.shape[1]
encoding_dim = 14
hidden_dim1 = int(encoding_dim / 2)
hidden_dim2 = 4
learning_rate = 1e-7
>input_layer = tf.keras.layers.Input(shape=(input_dim,))
encoder = tf.keras.layers.Dense(encoding_dim,activation="tanh",activity_regularizer =
tf.keras.regularizers.l2(learning_rate))(input_layer)
encoder = tf.keras.layers.Dropout(0.2)(encoder)
encoder = tf.keras.layers.Dense(hidden_dim1,activation='relu')(encoder)
encoder = tf.keras.layers.Dense(hidden_dim2,activation=tf.nn.leaky_relu)(encoder)
decoder = tf.keras.layers.Dense(hidden_dim1,activation='relu')(encoder)
decoder = tf.keras.layers.Dropout(0.2)(decoder)
decoder = tf.keras.layers.Dense(encoding_dim,activation='relu')(decoder)
decoder = tf.keras.layers.Dense(input_dim,activation='tanh')(decoder)
autoencoder = tf.keras.Model(inputs = input_layer,outputs = decoder)
autoencoder.summary()
>cp = tf.keras.callbacks.ModelCheckpoint(filepath="autoencoder_fraud.keras",mode='min',
  monitor='val_loss',verbose=2,save_best_only=True)
>early_stop = tf.keras.callbacks.EarlyStopping(monitor='val_loss',min_delta=0.0001,
            patience=10,verbose=11,mode='min',restore_best_weights=True)
autoencoder.compile(metrics=['accuracy'],loss= 'mean_squared_error',optimizer='adam')
>history = autoencoder.fit(normal_train_data,normal_train_data,epochs = nb_epoch,
       batch_size = batch_size,shuffle = True,validation_data =(test_data,test_data),
       verbose=1,callbacks = [cp,early_stop]).history
>plt.plot(history['loss'],linewidth = 2,label = 'Train')
plt.plot(history['val_loss'],linewidth = 2,label = 'Test')
plt.legend(loc='upper right')
plt.title('Model Loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.show()
>test_x_predictions = autoencoder.predict(test_data)
mse = np.mean(np.power(test_data - test_x_predictions, 2),axis = 1)
error_df = pd.DataFrame({'Reconstruction_error':mse,'True_class':test_labels})
>threshold_fixed = 50
groups = error_df.groupby('True_class')
fig,ax = plt.subplots()
```

```
for name,group in groups:
        ax.plot(group.index,group.Reconstruction_error,marker='o',ms = 3.5,linestyle='',
                label = "Fraud" if  name==1 else "Normal")
ax.hlines(threshold_fixed,ax.get_xlim()[0],ax.get_xlim()[1],colors="r",zorder=100,label="Threshold")
ax.legend()
plt.title("Reconstructions error for normal and fraud data")
plt.ylabel("Reconstruction error")
plt.xlabel("Data point index")
plt.show()
>import seaborn as sns
threshold_fixed = 52
pred_y = [1 if e > threshold_fixed else 0
          for e in
        error_df.Reconstruction_error.values]
error_df['pred'] = pred_y
conf_matrix = confusion_matrix(error_df.True_class,pred_y)
plt.figure(figsize = (4,4))
sns.heatmap(conf_matrix,xticklabels = LABELS,yticklabels = LABELS,annot = True,fmt="d")
plt.title("Confusion matrix")
plt.ylabel("True class")
plt.xlabel("Predicted class")
plt.show()
print("Accuracy :",accuracy_score(error_df['True_class'],error_df['pred']))
print("Recall :",recall_score(error_df['True_class'],error_df['pred']))
print("Precision :",precision_score(error_df['True_class'],error_df['pred']))
```

Practical : 5

```
>import matplotlib.pyplot as plt
import matplotlib as mpl
import matplotlib.pylab as pylab
import numpy as np
%matplotlib inline
>import re
>sentences = """Throughout my career, I have had the opportunity to work on a diverse range of
mobile app projects. My proficiency in Flutter, along with my Android development skills, has
allowed me to create user-friendly and high-performance applications. I take pride in my ability to
transform creative concepts into functional, feature-rich apps that deliver exceptional user
experiences."""
>sentences = re.sub('[^A-Za-z0-9]+', ' ', sentences)
sentences = re.sub(r'(?:^| )\w(?:$| )', ' ', sentences).strip()
sentences = sentences.lower()
>words = sentences.split()
vocab = set(words)
>vocab
>vocab_size = len(vocab)
embed_dim = 10
context_size = 2
>word_to_ix = {word: i for i, word in enumerate(vocab)}
ix_to_word = {i: word for i, word in enumerate(vocab)}
>data = []
for i in range(2, len(words) - 2):
  context = [words[i - 2], words[i - 1], words[i + 1], words[i + 2]]
  target = words[i]
  data.append((context, target))
print(data[:5])
>embeddings =  np.random.random_sample((vocab_size, embed_dim))
>def linear(m, theta):
  w = theta
  return m.dot(w)
>def log_softmax(x):
  e_x = np.exp(x - np.max(x))
  return np.log(e_x / e_x.sum())
>def NLLLoss(logs, targets):
  out = logs[range(len(targets)), targets]
  return -out.sum()/len(out)
>def log_softmax_crossentropy_with_logits(logits,target):
  out = np.zeros_like(logits)
  out[np.arange(len(logits)),target] = 1
  softmax = np.exp(logits) / np.exp(logits).sum(axis=-1,keepdims=True)
  return (- out + softmax) / logits.shape[0]
>def forward(context_idxs, theta):
  m = embeddings[context_idxs].reshape(1, -1)
  n = linear(m, theta)
  o = log_softmax(n)
  return m, n, o
>def backward(preds, theta, target_idxs):
  m, n, o = preds
```

```
    dlog = log_softmax_crossentropy_with_logits(n, target_idxs)
    dw = m.T.dot(dlog)
    return dw
>def optimize(theta, grad, lr=0.03):
    theta -= grad * lr
    return theta
>theta = np.random.uniform(-1, 1, (2 * context_size * embed_dim, vocab_size))
>epoch_losses = {}
for epoch in range(80):
    losses =  []
    for context, target in data:
        context_idxs = np.array([word_to_ix[w] for w in context])
        preds = forward(context_idxs, theta)
        target_idxs = np.array([word_to_ix[target]])
        loss = NLLLoss(preds[-1], target_idxs)
        losses.append(loss)
        grad = backward(preds, theta, target_idxs)
        theta = optimize(theta, grad, lr=0.03)
    epoch_losses[epoch] = losses
>ix = np.arange(0,80)
fig = plt.figure()
fig.suptitle('Epoch/Losses', fontsize=20)
plt.plot(ix,[epoch_losses[i][0] for i in ix])
plt.xlabel('Epochs', fontsize=12)
plt.ylabel('Losses', fontsize=12)
>def predict(words):
    context_idxs = np.array([word_to_ix[w] for w in words])
    preds = forward(context_idxs, theta)
    word = ix_to_word[np.argmax(preds[-1])]
    return word
>predict(['transform','creative','into','functional' ])
>def accuracy():
    wrong = 0
    for context, target in data:
        if(predict(context) != target):
            wrong += 1
    return (1 - (wrong / len(data)))
>accuracy()
```

Program:6

```
>import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Flatten
from tensorflow.keras.optimizers import Adam
from tensorflow.keras.preprocessing.image import ImageDataGenerator
import numpy as np
>train_dir = "C:/Users/salve/OneDrive/Desktop/cifar-10-img/cifar-10-img/train"
test_dir = "C:/Users/salve/OneDrive/Desktop/cifar-10-img/cifar-10-img/test"
>train_dir
>test_dir
>train_datagen = ImageDataGenerator(
   rescale=1.0 / 255,
)
test_datagen = ImageDataGenerator(
   rescale=1.0 / 255,
)
>train_batch_size = 5000
train_generator = train_datagen.flow_from_directory(
   train_dir,
   target_size=(32, 32),
   batch_size=train_batch_size,
   class_mode='categorical'
)
test_batch_size = 1000
test_generator = test_datagen.flow_from_directory(
   test_dir,
   target_size=(32, 32),
   batch_size=test_batch_size,
   class_mode='categorical'
)
>x_train, y_train =  train_generator[0]
x_test, y_test = test_generator[0]
print(len(x_train))
print(len(x_test))
>weights_path = "C:/Users/salve/Downloads/vgg16_weights_tf_dim_ordering_tf_kernels_notop.h5"
base_model = VGG16(weights=weights_path, include_top=False, input_shape=(32, 32, 3))
>for layer in base_model.layers:
  layer.trainable = False
>x = Flatten()(base_model.output)
x = Dense(256, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
x = Dense(256, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
predictions = Dense(10, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
```

```
model.compile(optimizer="adam", loss='categorical_crossentropy', metrics=['accuracy'])
>model.fit(x_train, y_train, batch_size=64, epochs=10, validation_data=(x_test, y_test))
>base_model = VGG16(weights=weights_path, include_top=False, input_shape=(32, 32, 3))
for layer in base_model.layers:
  layer.trainable = False
for layer in base_model.layers[len(base_model.layers) - 4:]:
  layer.trainable = True
x = Flatten()(base_model.output)
x = Dense(256, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
x = Dense(512, activation='relu')(x)
x = tf.keras.layers.Dropout(0.3)(x)
predictions = Dense(10, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)
model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=64, epochs=10, validation_data=(x_test, y_test))
>import matplotlib.pyplot as plt
predicted_value = model.predict(x_test)
>labels = list(test_generator.class_indices.keys())
>n=945
plt.imshow(x_test[n])
print("Preditcted: ",labels[np.argmax(predicted_value[n])])
print("Actual: ", labels[np.argmax(y_test[n])])
>n=9
plt.imshow(x_test[n])
print("Preditcted: ",labels[np.argmax(predicted_value[n])])
print("Actual: ", labels[np.argmax(y_test[n])])
>n=5
plt.imshow(x_test[n])
print("Preditcted: ",labels[np.argmax(predicted_value[n])])
print("Actual: ", labels[np.argmax(y_test[n])])
```